Programming Project #4: Image-Based Lighting
CS445: Computational Photography

Due Date: 11:59pm on Tuesday, Oct. 29, 2019

Overview

The goal of this project is to familiarize yourself high dynamic range (HDR) imaging, image based lighting (IBL), and their applications. By the end of this project, you will be able to create HDR images from sequences of low dynamic range (LDR) images and also learn how to composite 3D models seamlessly into photographs using image-based lighting techniques. HDR tonemapping can also be investigated as bells and whistles.

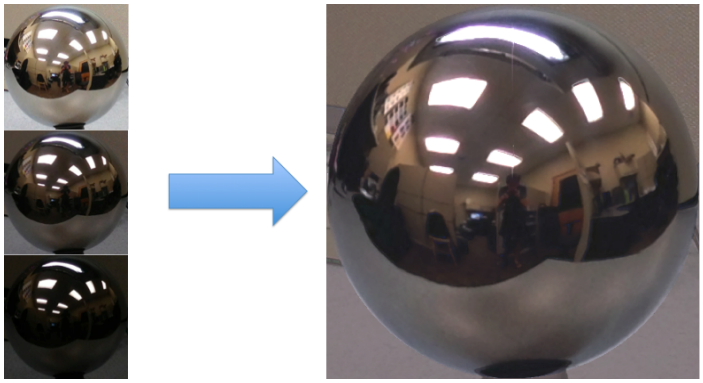
HDR photography is the method of capturing photographs containing a greater dynamic range than what normal photographs contain (i.e. they store pixel values outside of the standard LDR range of 0-255 and contain higher precision). Most methods for creating HDR images involve the process of merging multiple LDR images at varying exposures, which is what you will do in this project.

HDR images are widely used by graphics and visual effects artists for a variety of applications, such as contrast

enhancement, hyper-realistic art, post-process intensity adjustments, and image-based lighting. We will focus on their use in image-based lighting, specifically relighting virtual objects. One way to relight an object is to capture an 360 degree panoramic (omnidirectional) HDR photograph of a scene, which provides lighting information from all angles incident to the camera (hence the term image-based lighting). Capturing such an image is difficult with standard cameras, because it requires both panoramic image stitching (which you will see in project 5) and LDR to HDR conversion. An easier alternative is to capture an HDR photograph of a spherical mirror, which provides the same omni-directional lighting information (up to some physical limitations dependent on sphere size and camera resolution). We will take the spherical mirror approach, inspired primarily by Debevec's paper. With this panoramic HDR image, we can then relight 3D models and composite them seamlessly into photographs. This is a very quick method for inserting computer graphics models seamlessly into photographs and videos; much faster and more accurate than manually "photoshopping" objects into the photo.

Recovering HDR Radiance Maps (40 pts)

To the right are three pictures taken with different exposure times (1/24s, 1/60s, 1/120s) of a spherical mirror in my office. The rightmost image shows the HDR result (tonemapped for display). In this part of the project, you'll be creating your own HDR images. First, you need to collect the data.



What you need:

- Spherical mirror (see Piazza post)
- Camera with exposure time control. This is available on all DSLRs and most point-and-shoots, and even possible with most mobile devices using the right app; e.g. ProCamera on iOS, Camera FV-5 Lite on Android (this even has auto exposure bracketing, AEB). Automatic exposure bracketing is helpful but not really needed.
- Tripod / rigid surface to hold camera / *very* stead hand (not recommended)

Data collection (10 points)

1. Find a good scene to photograph. The scene should have a flat surface to place your spherical mirror on (see my example below). Either indoors or outdoors will work.
2. Find a fixed, rigid spot to place your camera. A tripod is best, but you can get away with less. I used the back of a chair to steady my phone when taking my images.
3. Place your spherical mirror on a flat surface, and make sure it doesn't roll by placing a cloth/bottle cap/etc under it. Make sure the sphere is not too far away from the camera -- it should occupy at least a 256x256 block of pixels.
4. Photograph the spherical mirror using at least three different exposure times. Make sure the camera does not move too much (slight variations are OK, but the viewpoint should generally be fixed). For best results, your exposure times should be at least 4 times longer and 4 times shorter (±2 stops) than your mid-level exposure (e.g. if your mid-level exposure time is 1/40s, then you should have at least exposure timess of 1/10s and 1/160s; the greater the range the better). Make sure to record the exposure times.
5. Remove the mirror from the scene, and from the same viewpoint as the other photographs, take another picture of the scene at a normal exposure level (most pixels are neither over- or under-exposed). This will be the image that you will use for object insertion/compositing (the "background" image).
6. After you copy all of the images from your camera/phone to your computer, load the spherical mirror images (from step 4) into your favorite image editor and crop them down to contain only the sphere (see example below).
7. Small alignment errors may occur (due to camera motion or cropping). One way to fix these is through various alignment procedures, but for this project, we won't worry about these errors. If there are substantial differences in camera position/rotation among the set of images, re-take the photographs.



From left to right: one of my sphere pictures (step 4), cropped sphere (step 6), empty scene (step 5)

Naive LDR merging (10 points)

After collecting data, load the cropped images, and resize them to all be square and the same dimensions (e.g. cv2.resize(ldr,(N,N)) N is the new size). Either find the exposure times using the EXIF data (usually accessible in the image properties), or refer to your recorded exposure times. To put the images in the same intensity domain, divide each by its exposure time (e.g. ldr1_scaled = ldr1 / exposure_time1). After this conversion, all pixels will be scaled to their approximate value if they had been exposed for 1s.
The easiest way to convert your scaled LDR images to an HDR is simply to average them. Create one of these for comparison to your later results.
To save the HDR image, use given write_hdr_image function. To visualize HDR image, use given display_hdr_image function.

LDR merging without under- and over-exposed regions (5 points)

The naive method has an obvious limitation: if any pixels are under- or over-exposed, the result will contain clipped (and thus incorrect) information. A simple fix is to find these regions (e.g. a pixel might be considered over exposed if its value is less than 0.02 or greater than 0.98, assuming [0,1] images), and exclude them from the averaging process. Another way to think about this is that the naive method is extended using a weighted averaging procedure, where weights are 0 if the pixel is over/under-exposed, and 1 otherwise. Note that with this method, it might be the case that for a given pixel it is never properly exposed (i.e. always either above or below the threshold in each exposure).

There are perhaps better methods that achieve similar results but don't require a binary weighting. For example, we could create a weighting function that is small if the input (pixel value) is small or large, and large otherwise,

and use this to produce an HDR image. In python, such a function can be created with: w = lambda z: float(128-abs(z-128)) assuming pixel values range in [0,255].

LDR merging and response function estimation (15 points)

Nearly all cameras apply a non-linear function to recorded raw pixel values in order to better simulate human vision. In other words, the light incoming to the camera (radiance) is recorded by the sensor, and then mapped to a new value by this function. This function is called the film response function, and in order to convert pixel values to true radiance values, we need to estimate this response function. Typically the response function is hard to estimate, but since we have multiple observations at each pixel at different exposures, we can do a reasonable job up to a missing constant.

The method we will use to estimate the response function is outlined in this paper. Given pixel values Z at varying exposure times t, the goal is to solve for g(Z) = ln(R*t) = ln(R)+ln(t). This boils down to solving for R (irradiance) since all other variables are known. By these definitions, g is the inverse, log response function. The paper provides code to solve for g given a set of pixels at varying exposures (we also provid gsolve function in our utils folder). Use this code to estimate g for each image channel (r/g/b). Then, recover the HDR image using equation 6 in the paper.
Some hints on using gsolve:

- When providing input to gsolve, don't use all available pixels, otherwise you will likely run out of memory / have very slow run times. To overcome, just randomly sample a set of pixels (100 or so can suffice), but make sure all pixel locations are the same for each exposure.
- The weighting function w should be implemented using Eq. 4 from the paper (this is the same function that can be used for the previous LDR merging method, i.e. w = lambda z: float(128-abs(z-128)).
- Try different lambda values for recovering g. Try lambda=1 initially, then solve for g and plot it. It should be smooth and continuously increasing. If lambda is too small, g will be bumpy.
- Refer to Eq. 6 in the paper for using g and combining all of your exposures into a final image. Note that this produces log radiance values, so make sure to exponentiate the result and save absolute radiance.

Panoramic transformations (20 pts)

Now that we have an HDR image of the spherical mirror, we'd like to use it for relighting (i.e. image-based lighting). However, many programs don't accept the "mirror ball" format, so we need to convert it to a different 360 degree, panoramic format (there is a nice overview of many of these formats here). For this part of the project, you should implement the mirror ball to equirectangular (latitude longitude) transformation. Most rendering software accepts this format, including Blender's Cycles renderer, which is what we'll use in the next part of the project.

To perform the transformation, you need to figure out the mapping between the mirrored sphere domain and the equirectangular domain. Hint: calculate the normals of the sphere (N) and assume the viewing direction (V) is constant. You can calculate reflection vectors with R = V - 2 * dot(V,N) * N, (NOTE that you'd have to implement channel-wise dot product) which is the direction that light is incoming from the world to the camera after bouncing off the sphere. The reflection vectors can then be converted to, providing the latitude and longitude (phi and theta) of the given pixel (fixing the distance to the origin, r, to be 1). Note that this assumes an orthographic camera (which is a close approximation as long as the sphere isn't too close to the camera).

Next, the equirectangular domain can be created by making an image in which the rows correspond to theta and columns correspond to phi in spherical coordinates, e.g,

```
EH, EW = 360, 720
phi_fst_half = np.arange(math.pi, 2*math.pi, math.pi / (EW // 2))
phi_snd_half = np.arange(0 * math.pi, math.pi, math.pi / (EW // 2))
theta_range = np.arange(0, math.pi, math.pi / EH)
```

```
phi_ranges = np.concatenate((phi_fst_half, phi_snd_half))
phis, thetas = np.meshgrid(phi_ranges, theta_range)
```

Note that by choosing 360 as EH and 720 as EW, we are making every pixel in equirectangular image to correspond to area occupied by 0.5 degree x 0.5 degree in spherical coordinate. Now that you have the phi/theta for both the mirror ball image and the equirectangular domain, use scipy's scipy.interpolate.griddata function to perform the transformation. Below is an example transformation.

Note that the following portion of the project depends on successfully converting your mirror ball HDR image to the equirectangular domain. If you cannot get this working, you can request code from the instructors at a 20 point penalty (i.e. no points will be awarded for this section, but you can do the later sections).

Rendering synthetic objects into photographs (30 pts)

Next, we will use our equirectangular HDR image as an image-based light, and insert 3D objects into the scene. This consists of 3 main parts: modeling the scene, rendering, and compositing. Specific instructions follow below; if interested, see additional details in Debevec's paper.

Begin by downloading/installing Blender here. Note that this part of tutorial assumes that you are using blender version 2.79. If you want to use newest blender 2.8 or above, please refer to this page for step by step reference. In the course materials package below, locate the blend file under samples directory and open it. This is the blend file I used to create the result at the top of the page. The instructions below assume you will modify this file to create your own composite result, but feel free to create your own blend file from scratch if you are comfortable with Blender.

Modeling the scene

To insert objects, we must have some idea of the geometry and surface properties of the scene, as well as the lighting information that we captured in previous stages. In this step, you will manually create rough scene geometry/materials using Blender.

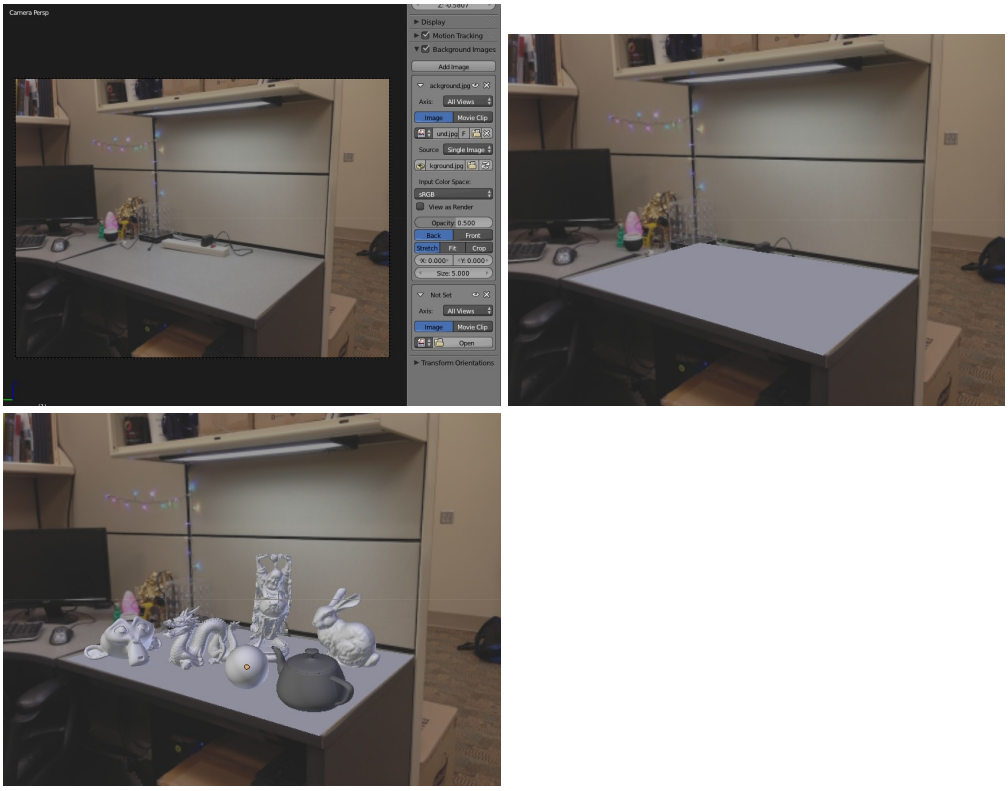
With the sample blend file open, add your background image to the scene. In the 3D view window near the bottom right, locate "Background Images". Make sure this is checked, and click "Add image", then click "Open" and locate your background image from step 4 of data collection. Make sure your view is from the camera's perspective by pressing View->Camera; you should see your image in view.

Next, model the "local scene." That is, add simple geometry (usually planes suffice) to recreate the geometry in the scene near where you'd like to insert objects. For best results, this should be close to where you placed the spherical mirror. Feel free to use the sample scene provided and move the vertices of the plane to match the surface you'd like to recreate (ignore the inserted bunny/teapot/etc for now). Once you're happy with the placement, add materials to the local scene: select a piece of local scene geometry, go to Properties->Materials, add a Diffuse BSDF material, and change the "Color" to roughly match the color from the photograph.

Then, add your HDR image (the equirectangular map made above) to the scene. First, use notebook to save the HDR panorama: write_hdr_image(eq_image, 'equirectangular.hdr'). In the Properties->World tab, make sure

Surface="Background" and Color="Environment Texture". Locate your saved HDR image in the filename field below "Environment Texture".

Finally, insert synthetic objects into the scene. Feel free to use the standard models that I've included in the sample blend file, or find your own (e.g. Turbosquid, Google 3D Warehouse, DModelz, etc). Add interesting materials to your inserted objects as well. Once finished, your scene should now look something like the right image below.



Blender scene after: loading background image, modeling local scene, inserting objects

Rendering

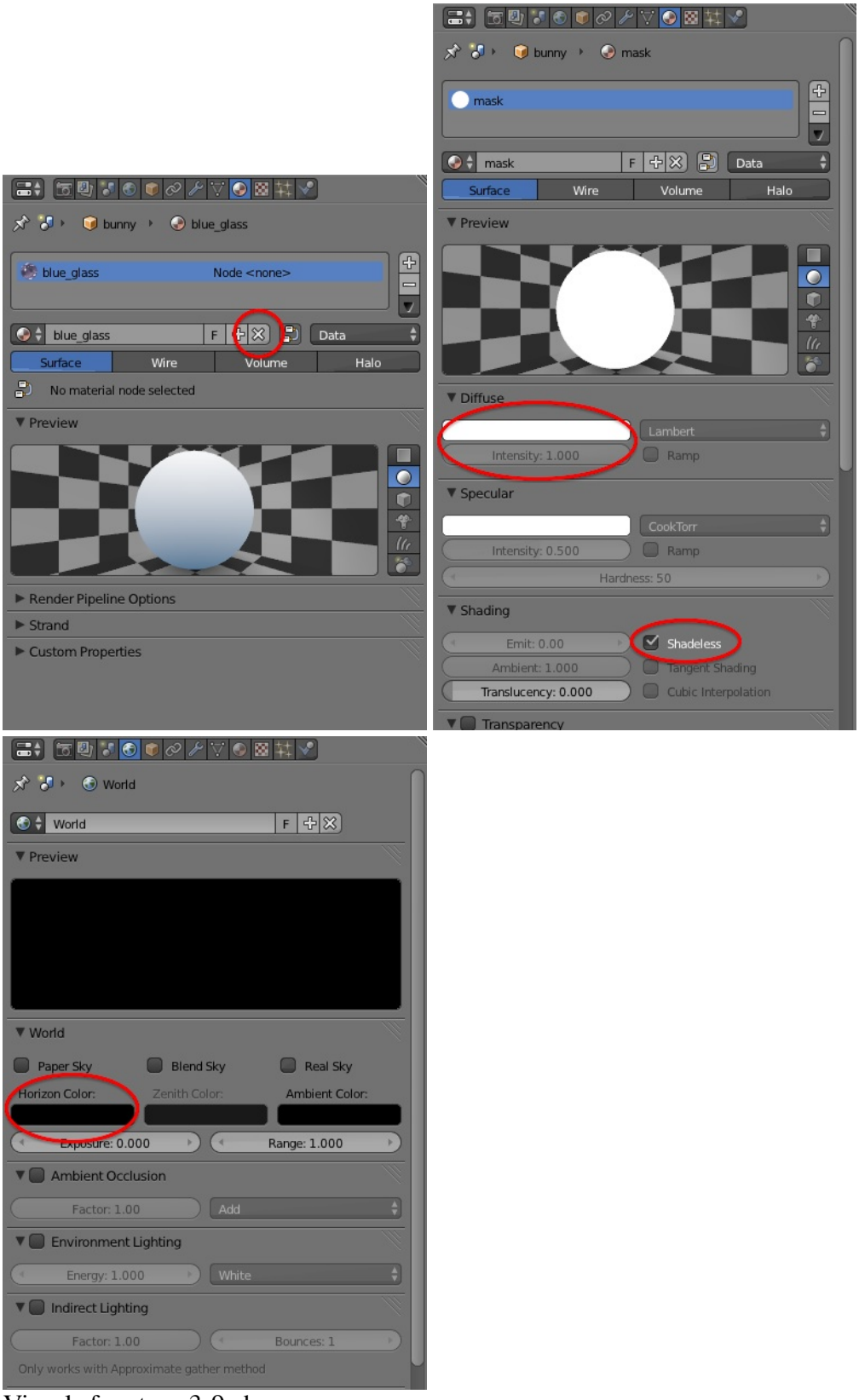
We can now render the scene to see a preview of what the inserted objects will look like. Make sure "Cycles Render" is selected at the top of Blender's interface, and then render the scene (F12). Your rendering might be too bright/dark, which is caused because we don't know the absolute scale of the lighting, so this must be set manually. To fix, adjust the light intensity (Properties->World tab, adjust "Strength" setting under "Color" accordingly). Once you're happy with the brightness, save the rendered result to disk.

My rendered scene is down below; as you can tell, this is not quite the final result. To seamlessly insert the objects, we need to follow the compositing procedure outlined by Debevec (Section 6 of the paper). This requires rendering the scene twice (both with and without the inserted objects), and creating an inserted object mask.

Next, we'll render the "empty" scene (without inserted objects). Create a copy of your blender scene and name it something like ibl-empty.blend. Open up the copy, and delete all of the inserted objects (but keep the local scene geometry). Render this scene and save the result.

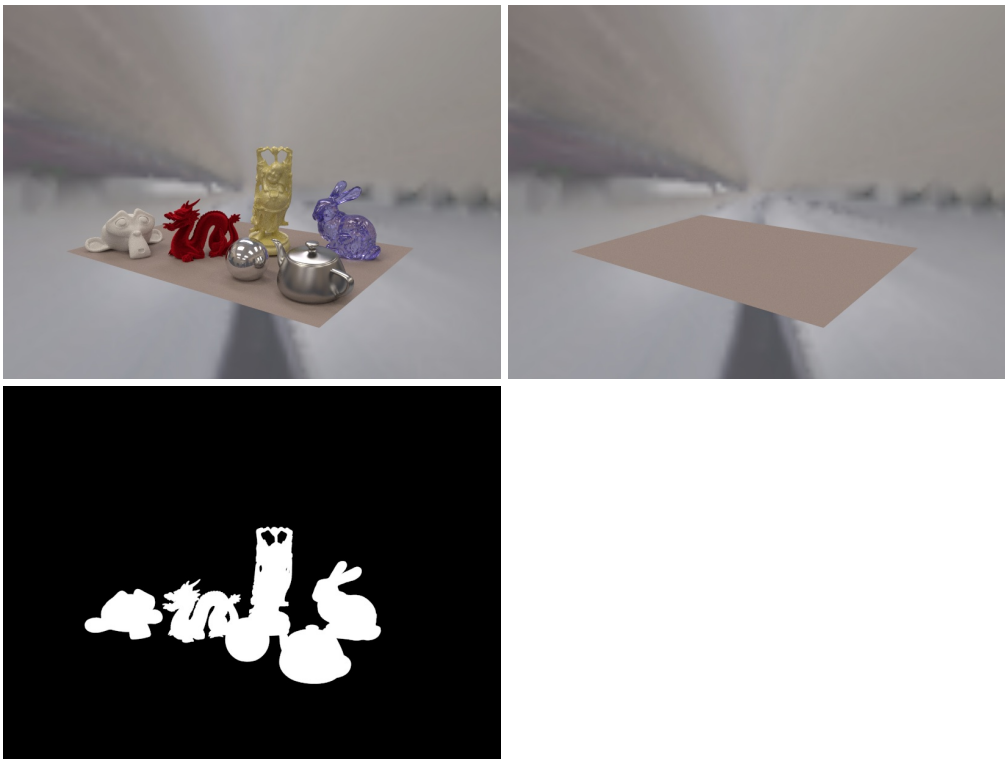
Finally, we need to create an object mask. The mask should be 0 where no inserted objects exist, and greater than 0 otherwise. First, create another duplicate of your scene and open it up (e.g. ibl-mask.blend). We can create the mask quickly using Blender by manipulating object materials and rendering properties:

1. In the top panel, make sure it says "Blender Render" (if it says something else, e.g. Cycles Render, change it to Blender Render)
2. Select an object (right click on it)
3. Go to the materials panel (Properties->Materials; looks like a black/red circle)
4. Remove the Material by pressing the 'x' (to the left of "Data")
5. Click "New" to add a new material
6. In the new material properties, under "Diffuse", change Intensity=1 and the RGB color = (1,1,1)
7. Under "Shading", check the "Shadeless" box
8. Repeat for all inserted objects
9. In Properties->World, set the Horizon RGB color = (0,0,0)
10. Render the scene and save your mask as a PNG (or some lossless format)



Visuals for steps 3-9 above.

After these steps, you should have the following three rendered images:



Rendered image with objects, rendering without objects, object mask

To simplify this process, we have created a script `ibl_script.py` for you, which is in the project materials. To use it, edit the `project_path_variable` and edit the call to `object_rendering_mode` to set the lighting strength and local surface color. Either the process above or this script are ok to use.

Compositing

To finish the insertion, we will use the above rendered images to perform "differential render" compositing. This can be done using a simple pixel-wise equation. Let R be the rendered image with objects, E be the rendered image without objects, M be the object mask, and I be the background image. The final composite is computed with:

composite = M*R + (1-M)*I + (1-M)*(R-E)*c

The first two terms effectively pastes the inserted objects into the background image, and the third term adds the lighting effects of the inserted objects (shadows, caustics, interreflected light, etc), modulated by c. Set c=1 initially, but try different values to get darker or lighter shadows/interreflections. The final compositing result I achieved using my image-based light is at the top of the page.

Some tips on using Blender

- Save your Blender file regularly, and always before closing (on some operating systems, Blender will close without prompting to save).
- To move more than one object at once, select multiple objects using shift. Pressing 'a' deselects all objects/vertices.
- You can edit vertices directly in "Edit mode" (tab toggles between Object and Edit modes).
- **For image-based lighting, the camera should always be pointed such that the +z axis is up, and the +x axis is forward** (as it is in the sample blend file in the project materials). This is the coordinate system used

by Blender when applying an image-based light to the scene; otherwise your IBL will have incorrect orientation w.r.t. the scene.
- You can however translate the camera rather than moving the objects in the scene (but make sure the rotation is fixed, as per the above bullet).

Bells & Whistles (Extra Points)

Other panoramic transformations (20 pts)
Different software accept different spherical HDR projections. In the main project, we've converted from the mirror ball format to the equirectangular format. There are also two other common formats: angular and vertical cross (examples here and here). Implement these transformations for 10 extra points each (20 possible).

Photographer/tripod removal (20 pts)
If you look closely at your mirror ball images, you'll notice that the photographer (you) and/or your tripod is visible, and probably occupies up a decent sized portion of the mirror's reflection. For 20 extra points, implement one of the following methods to remove the photographer: (a) cut out the photographer and use in-painting/hole-filling to fill in the hole with background pixels (similar to the bells and whistles from Project 2), or (b) use Debevec's method for removing the photographer (outlined here, steps 3-5; feel free to use Debevec's HDRShop for doing the panoramic rotations/blending). The second option works better, but requires you to create an HDR mirror ball image from two different viewpoints, and then merge them together using blending and panoramic rotations.

Local tonemapping operator (30 pts)
HDR images can also be used to create hyper-realistic and contrast enhanced LDR images. This paper describes a simple technique for increasing the contrast of images by using a local tonemapping operator, which effectively compresses the photo's dynamic range into a displayable format while still preserving detail and contrast. For 30 extra credit points, implement the method found in the paper and compare your results to other tonemapping operations (see example below for ideas). We provide you bilateral_filter function, but do not use any other third party code. You can find some example HDR images here, including the memorial church image used below.

From left to right: simple rescaling, rescaling+gamma correction, local tonemapping operator, local tonemapping+gamma correction.

Materials

- Starter material for the project: Includes notebook, util codes, LDR mirror ball pictures, the resulting equirectangular HDR image, and a sample blender file for compositing.

Deliverables

For the core part of this project, you are highly recommended to work with one partner (groups of 2 max). The two group members should be involved in every step (e.g., take photos together, write code together or each write all code in parallel, figure out blender together). You should both submit code and a project page, but they can be duplicates. Any bells and whistles must be done on your own with independent results. In your compass submission text, include the name and netid of your partner and a link to his or her web page as well. If the pages are exact duplicates (worth up to 100 points), please say so to save the grader some checking.

You know the drill: create a web page and thumbnail, and submit code/text/link on Compass. See project instructions for details. As detailed above, also include your partner's name, netid, and project address in the text.

Use both words and images to show us what you've done. Please:

- For all three LDR merging stages (naive, accounting for under- and over-exposure, and with response function estimation), show the estimated log irradiance for each exposure, followed by the resulting merged, HDR log irradiance. To bring your results into a displayable range, linearly scale the images to the range [0,1] (subtract the min and divide by max-min), and use the same scaling for each stage (i.e. compute the max and min over all exposures for a particular merging stage). For each stage, should the irradiances look the same? Why or why not? To help you answer, examine a few pixels across the various exposures. Add these answers to the description/discussion for your LDR-to-HDR merging results. (Remember that g(Z) = ln(t) + ln(E), where Z are pixel values and t is the exposure time in seconds. For the first two stages, we haven't estimated g, so you can assume ln(Z) = g(Z).)
- For the third LDR merging stage (response function estimation), plot the estimated function g; that is, plot pixel value vs g(pixel value) i.e. plt.plot(range(255), g).
- Show your equirectangular HDR image (again in a displayable range) and discuss your method for implementing the domain transformation.
- Show your background image and at least one compositing result, along with the intermediate renderings (render with objects, render without objects, object mask).
- Include notebook that contains code for merging the LDR images into an HDR image, and your domain transformation code. It should be clear from the function names which is which (e.g., makehdr_naive, makehdr_exposed, makehdr_gsolve, perspective_transform"). You do not need to submit any blend files used to create your results.
- Describe bells and whistles under a separate heading and include relevant code/results.

Scoring

The core assignment is worth **100** points, as follows:

- **40 points** for LDR-to-HDR merging (image collection=10, naive merging=10, accounting for under- and over-exposed regions=5, jointly calculating response function=15, including figures, plots and discussion as in the deliverables).
- **20 points** for the mirror ball to equirectangular domain transformation.
- **20 points** for the first image-based light compositing result (including intermediate renderings).
- **10 points** for an additional IBL result using either a new scene or new objects (must be downloaded and exclude the bunny, monkey, etc, which are included in project materials)
- **10 points** for quality of results and project page.

You can also earn up to **70 extra points** for the bells & whistles mentioned above (up to 20 for additional domain

transformations; 20 for photographer removal; 30 for implementing the local tonemapping operator of Durand and Dorsey).